Detecting Anomalous and Unknown Intrusions Against Programs in Real-Time
DARPA SBIR Phase I Final Report*
Sponsored by
Defense Advanced Research Projects Agency
(Information Technology Office)
ARPA Order Nr. D611, Amdt 11
Issued by U.S. Army Missile Command Under
Contract No. DAAH01-97-C-R095

Anup K. Ghosh, Gary McGraw, James Wanken, & Frank Charron
Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
(703)404-9293 (voice)
aghosh@rstcorp.com
http://www.rstcorp.com

Effective Date of Contract: 24FEB1997 Contract Expiration Date: 26SEP97

September 25, 1997

*THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESS OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Sept. 25, 1997	Final Sept. 25, 1997-Sept. 26, 1997

4. TITLE AND SUBTITLE

Detecting Anomalous and Unknown Intrusions Against Programs in Real-Time

5. FUNDING NUMBERS

DAAH01-97-C-R095

6. AUTHOR(S)

Anup Ghosh, Gary McGraw, James Wanken, Frank Charron

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Project Agency
(Information Technology Office)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARPA Order Nr. D611,
Amdt 11

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report discusses the research and results discovered under a Phase I SBIR program awarded by DARPA and the U.S. Missile Command contract number DAAH01-97-C-R095. The main objective of this Phase I research grant is to study the feasibility in using connectionist approaches to detecting the existence of anomalous or unknown intrusions against programs in real-time. The research resulted in the development of a prototype that can be used to train a neural network on both normal and anomalous usage and behavior of programs. The prototype was applied to the usage of Web-based applications as well as to the usage and behavior of a system utility program. Initial results demonstrate the viability of this approach to detecting unknown attacks against systems through misuse and anomalous behavior of software programs. In addition to presenting the empirical results, we discuss theoretical issues in the constraints of this approach, as well as the commercial potential we see in this approach. Though many avenues of research, development, and commercialization still exist, the initial results from this Phase I project demonstrate the feasibility of using connectionist networks to detecting anomalous usage and behavior in programs.

14. SUBJECT TERMS	15. NUMBER OF PAGES 28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclass	unclass	unclass	

Abstract

This report discusses the research and results discovered under a Phase I SBIR program awarded by DARPA and the U.S. Army Missile Command contract number DAAH01-97-C-R095. The main objective of this Phase I research grant is to study the feasibility in using connectionist approaches to detecting the existence of anomalous or unknown intrusions against programs in real-time. The research resulted in the development of a prototype that can be used to train a neural network on both normal and anomalous usage and behavior of programs. The prototype was applied to the usage of Web-based applications as well as to the usage and behavior of a system utility program. Initial results demonstrate the viability of this approach to detecting unknown attacks against systems through misuse and anomalous behavior of software programs. In addition to presenting the empirical results, we discuss theoretical issues in the constraints of this approach, as well as the commercial potential we see in this approach. Though many avenues of research, development, and commercialization still exist, the initial results from this Phase I project demonstrate the feasibility of using connectionist networks to detecting anomalous usage and behavior in programs.

1 Intrusion Detection

As the current trend toward networked computers increases, so too does the security issues of such systems. Not only are attacks becoming more frequent, but the size and complexity of networked systems are overwhelming security administrators. Intrusion detection techniques have begun to become widely researched and implemented in an attempt to grapple with the increasing number of Internet-based attacks.

Intrusion detection can be roughly defined as a method to detect any unauthorized use of a system. An intrusion is most commonly used to refer to as an attack from the "outside" that gains access to the "inside" system, and then uses this as a base to illegally use resources, view confidential material, or even try to pervert or harm the system. The term is less commonly used, yet equally applicable, to describe attacks from within the system by people who have valid accounts. This type of attack involves misuse of account holders' privileges to view documents or to perform operations, that although are permitted, are in violation of the organization's security policy. Examples of intrusive activities could be: exploitation of hidden capabilities or software programming flaws in a program that allows unexpected use of the application, abuse of permissions or access, and failure of authentication procedures.

Intrusion detection techniques fall into two main paradigms: misuse detection and anomaly detection. Misuse detection methods attempt to model attacks on a system as specific patterns, then systematically scan the system for occurrences of these patterns. This process involves a specific encoding of previous behaviors and actions that were deemed intrusive or malicious. Anomaly detection assumes that intrusions are highly correlated to abnormal behavior exhibited by either a user or an application. The basic idea is to baseline normal behavior of the object being monitored and then flag behaviors that are significantly different from this baseline as abnormalities, or possible intrusions. The research project funded under this SBIR grant utilizes the anomaly detection approach discussed in Section 3.

2 Misuse Detection

Since specific attack sequences are encoded into misuse detection systems, it is very easy to determine exactly which attacks, or possible attacks, the system is currently experiencing. This methodology can even be used in an attempt to predict the next action that the intruder will take. In theory, these systems can be encoded with arbitrarily complex attack patterns, although, in practice it is difficult to efficiently encode and scan for very complex patterns. Misuse detection schemes also allow a great deal of flexibility as many systems permit additional attack patterns to be added with only a minimal amount of change to the existing system.

However, there are distinct drawbacks to misuse detection schemes. The most prominent of these being their extreme dependence on the people who devise the patterns. Since these systems only scan for encoded patterns, their detection ability is limited by the knowledge of the people who encode the patterns and thus will not be able to detect novel or unknown attacks. Another drawback of this approach is the sheer mass of data which modern processors create that have to be matched. Usually, scanning the entire system state cannot be performed and a subset of the audit trail for the system must be used to match against.

The strength of misuse detection systems is the exactness that can be obtained in defining malicious patterns or activities, but their shortcoming is that they are susceptible to very elaborate attacks.

2.1 Approaches to Misuse Detection

There are four major approaches to misuse detection in the current literature [Kumar and Spafford, 1996].

Expert Systems - explicit knowledge about the patterns are encoded as `if-then` rules.

Model Based Reasoning - uses models of misuse and evidential reasoning to determine if current activities represent intrusive behaviors.

State Transition Analysis - implements the detection scheme as a sequence of state transitions.

Keystroke Monitor - patterns in the key stroke history are directly used to detect intrusive behavior.

2.1.1 Expert Systems

Expert system implementations attempt to encode knowledge about possible intrusive behavior as a set of `if-then` rules. It is difficult to encode a specific sequence of actions efficiently in this scheme since clauses in the `if-then` rules are inherently unordered. Thus, most rules will fire due to an arbitrary permutation of its clauses [Kumar and Spafford, 1996]. Unfortunately, the quality of the rules is highly dependent on the people who devise them, and the system is very susceptible to novel attacks. The main advantage achieved by encoding the patterns into `if-then` rules is that the reasoning is completely separated from the formulation of the specific problem domain. Another advantage is that the rules are not necessarily tied to an individual's or system's past behavior. This can eliminate the possible scenario where a user can subvert the system by slowly changing their behaviors.

2.1.2 Model Based Reasoning

A model based reasoning system keeps a list of possible attacks currently being experienced by the system. Then, the system attempts to anticipate the next actions the intruder will take and eliminates candidate attacks as further states add or subtract evidence for particular scenarios. This method is based on a sound theory of reasoning in the presence of uncertainty which is not easily obtainable in many other systems such as an expert system. Furthermore, it may save processing as time elapses since fewer and fewer possible intrusion scenarios are being considered. Thus, model based systems react to the situation, filtering out unnecessary data and concentrating on the data most likely to produce an intrusion [Kumar and Spafford, 1996]. Instructive descriptions of the attack can also be obtained, as well as a list of possible actions the intruder will take next based on the current intrusion models being considered. If the number of system intrusions is low in comparison to the volume of audit data, the amount of data that needs to be processed can quickly swamp the system. This will result

from the fact that the system will have many possible intrusions in consideration, most of which are false leads [Lunt, 1993]. SRI is also researching a model based approach to intrusion detection [Garvey and Lunt, 1991].

2.1.3 State Transition Analysis

State transition analysis approaches model attacks as a sequence of state transitions of the system. States are connected by arcs representing events with associated Boolean conditions that must be satisfied to transition to the next state [Kumar and Spafford, 1996]. This approach has been pursued by Ilgun [Ilgun, 1992] and Porras and Kemmerer [Porras and Kemmerer, 1992].

2.1.4 Keystroke Monitor

Keystroke monitoring attempts to match patterns of misuse to specific keystroke sequences. This method is plagued by the fact that general and consistent availability of keystrokes is difficult to obtain. Also, there is a large number of ways to execute a specific attack on a system at the keystroke level. Use of command aliases is just one way to effectively defeat this approach [Kumar and Spafford, 1996].

3 Anomaly Detection

The primary advantage of anomaly detection is its ability to detect novel and unknown intrusions. This is possible because anomaly detection does not scan for specific patterns, but instead compares current activities against previous statistical models of past program behavior. Any activity sufficiently deviant from the model will be flagged as anomalous, and hence considered as a possible attack. Furthermore, anomaly detection schemes are based on actual user histories and system data to create its internal models rather than predefined patterns.

One clear drawback of anomaly detection is its inability to identify the specific type of attack that is occurring. It is also difficult to capture some well known intrusion methods in the typical statistical models used. Anomaly detection schemes can also be subverted if users know that they are being monitored. Once attackers knows that they are being monitored, they can potentially slowly pervert the system in a manner that will eventually allow them to perform some form of malicious activity. For example, in systems which compare a user's behavior to previous profiles, the user could slowly change their behaviors until they establish a behavior pattern from where they can mount an attack. However, building new statistical profiles or modifying the profiles in some way usually takes a large amount of time and data.

3.1 Approaches to Anomaly Detection

Anomaly detection schemes found in the literature can be classified as follows:

Expert Systems - profiles of observed user behavior are constructed and updated,

Statistical Modeling - statistical profiles of user or program behavior are constructed,

Neural Networks - statistical profiles of user behavior are modeled using a neural network,

Immunological Approach - detection technique is a modeled from T-cell behavior.

3.1.1 Expert Systems

Expert systems have also been used in the construction of anomaly detection systems. These systems construct user profiles of observed user behavior and they also often adaptively learn and update these profiles. Because these profiles are learned, and are not pre-defined, these systems have the potential to find behaviors that human experts may have overlooked or never considered. Providing system security based on a user's profile though, makes the method highly dependent on the variability in each individual's behavior. If users have very erratic behaviors, the system will not be able to easily distinguish between normal and abnormal behavior. While when a user has a very stable behavior, possible intrusions should be much easier to flag. This is one of the approaches incorporated into IDES (Intrusion Detection Expert System) [Lunt, 1993, Lunt and Jagannathan, 1988, Lunt, 1990, Lunt et al., 1992].

3.1.2 Statistical Modeling

Once statistical models of user behavior have been constructed, they can serve to delimit normal versus abnormal behavior. Building statistical user profiles for baselines is very difficult since user behavior can be very complex. Invalid assumptions on the distributions of these attributes can greatly affect the performance and ability of the system. Since having distinct profiles for each individual user can become quite cumbersome, methods for breaking up users into profile classes have been attempted such that the baseline statistics are composed for the entire class [Lunt, 1993]. Another possible methodology could be to profile programs or files instead of individual users. Statistics could be based upon characteristics such as CPU time, files accessed, or inputs and outputs of programs [Lunt, 1993].

3.1.3 Neural Networks

Neural networks have also been used as another way of implementing the statistical modeling approach without actually constructing the statistical models. Neural networks provide the great advantage that they do not assume any of the underlying statistical properties of the modeled distribution. Development and implementation of complicated statistical analysis of such systems is also very cost prohibitive. Neural networks are relatively easy to modify if they are moved to a different working environment where the users have varying behaviors. Neural networks, though, provide no information about the reasoning methodology behind their processing. Thus, given that a set of actions was flagged to be anomalous by the system, it is very difficult to determine what the neural network believed to be the anomalous aspect [Lunt, 1993].

3.1.4 Immunological Approach

The immunological approach was inspired by the operation of T-cells in the immune system. T-cells with randomly generated receptors are first created. Then, any T-cell whose receptor

matches a self-protein is not released into the rest of the body. Thus, this provides a way to recognize changes to cells, or cells that are not part of the self-protein set. The security method distinguishes self-strings, which represent protected data or activities, from non-self–strings, which represent foreign or malicious data or activities. This is accomplished by generating detectors for anything that is not in the set of self-strings. This method can withstand change-detection problems which contain noise, or involve dynamic strings of data, such as running processes' activity patterns. But this generality also constrains this approach so it may not be as efficient as a knowledge-intensive special-purpose implementation. The process of generating the detectors can be very time consuming, and it may still leave some holes in the state space which no detector can cover [D'haeseleer et al., 1996].

4 Neural Network Discussion

Neural network paradigms are quite numerous. In order to understand neural networks in general, a broad, yet unambiguous, taxonomy needs to be constructed. In this discussion, neural networks will be divided into groups with respect to their particular learning and recall methods.

4.1 Learning Paradigms

Aside from the architecture of the network, the learning method is one of the most important aspects of the neural network. The learning process is what creates a specific mapping from the input vectors to the output vectors. Through the process of training, the learning method allows a general neural net to adapt to solve a particular problem. There are two main learning paradigms, each with numerous algorithms [Pal and Srimani, 1996].

4.1.1 Supervised learning

In supervised learning, for each input the desired neural network output must also be supplied to the neural net. The weights are then updated in such a manner as to allow the network to produce outputs that are as close to the desired outputs as possible.

4.1.2 Unsupervised learning

In unsupervised learning, the desired neural network output values do not need to be supplied. The network tries to group, or cluster, the input sequences into categories based on the statistical regularities and the underlying structure of the inputs.

4.2 Recall Methods

There are essentially two recall methods used in neural networks: feedforward and feedback recall.

In neural networks that use feedforward recall, the inputs are applied to the network and the activations of each neuron cascade forward layer by layer to the output nodes of the network. This type of recall is termed instantaneous because once an input has been

applied the outputs will be valid after a fixed duration on time (dependent on factors such as network size, activation function used, connection scheme, etc.). The time at which the outputs of the network become valid should be independent of the specific inputs applied to the network.

In networks that employ feedback recall, once inputs have been applied to the network the network must repeatedly cycle through its recall process until the outputs have become stable. Feedback recall usually occurs when there is a set of connections that create a cycle in the network. These networks are not considered instantaneous as different inputs may take varying amounts of time until the outputs have become stable.

4.3 Supervised Learning and Feedforward Recall

This class of networks is the main-stay of neural networks. They combine a fairly straightforward encoding and recall scheme with a fairly simplistic network topology.

4.3.1 Backpropagation

The backpropagation, or backprop, network is probably the most used neural network. The standard architecture consists of an input layer, at least one hidden layer (neurons that are not directly connected to the inputs or outputs), and an output layer. Typically, there are no connections between neurons in the same layer or to a previous layer, though some implementations do have feedback.

Backprops have generalized capability, as such they can produce nearly correct outputs for inputs that were not used in the training set. In theory, no more than two hidden layers are needed in a neural network (for classification) since the network can generate arbitrarily complex regions in the state space [Lippmann, 1991]. One of the backprops' main drawbacks, though, is that it tends to be very computationally complex and is time consuming to train. These networks are well-suited for applications in classification, function approximation, and prediction [Jain et al., 1996].

The training cycle of a backprop proceeds in two distinct phases. First, the input is submitted to the network and the activations for each level of neurons is cascaded forward. In the learning phase, the desired output is compared with the network's output. If these vectors do not agree, the network updates the weights starting at the output neurons. Then, the change in weights are calculated for the previous layer. This process continues to cascade through the layers of neurons toward the input neurons, hence the name backpropagation.

4.4 Unsupervised Learning and Feedforward Recall

This class of networks is not used as often as the supervised variety, but there is still a fair number of standard implementations. There are a number of implementations analogous to the corresponding supervised feedforward recall networks such as the counterpropagation network [Simpson, 1990]. Typically, these algorithms attempt to train a single output to have a high value when a distinct input is presented, thus these networks are often used for classification. Other varieties of unsupervised feedforward networks are: the Learning Matrix, the Linear Associative Memory, and the Learning Vector Quantizer.

4.5 Unsupervised learning and Feedback Recall

This is another heavily used class of networks and has many derivations. Unsupervised feedback recall networks also have a range of very distinctive uses ranging from memories to clustering of data.

4.5.1 Hopfield Networks

Hopfield networks consists of a single layer of neurons each of which have inputs, outputs, and another set of connections to the inputs of all of the other neurons in the same layer. Classification proceeds in two stages. First, the input is clamped onto the network to initialize the network. Then, the input is removed and the network propagates these inputs through the network, and uses feed-back from the outputs to the inputs to start another cycle. These sequential iterations continue until the system reaches a stable condition (e.g., the network has fallen into one of the stored attractors of the network) [Zurada, 1992]. One problem with this method is that if multiple attractors have the same energy levels the network can become caught in a cycle between these equal energy states. Hopfield networks are usually used for problems that can be represented as an analogous optimization problem (e.g., the traveling salesman problem) [Jain et al., 1996].

4.5.2 Associative Memories

Associative, or auto-associative, memories are based on storing associations for pattern retrieval and restoration of an incomplete, or noisy input. The architecture for this type of network is varied and can be composed of a simple feed-forward network, a recurrent (Hopfield) network, or a network built from multiple recurrent networks. A set of exemplars is first stored in the memory. Next, during recall, a specific key input pattern containing a portion of the desired retrieved data is clamped to the network. The network then uses the stored associations to recall the stored exemplar. Retrieval may provide the correct exemplar, an incorrect exemplar, or it may even produce an invalid exemplar that was not stored in the network at all. Associative memories can be applied to such problems as content-addressable storage, image restoration, classification, and code correcting [Zurada, 1992].

4.5.3 Adaptive Resonance Theory

The ART network, also referred to as a Carpenter/Grossberg classifier, attempts to cluster the input data as noted above. A unique property of ART networks is that a pre-defined number of clusters is not needed, only an upper bound on the number of possible clusters. The network has the capability to classify up to N groups, but the outputs are not used until the network decides a new cluster should be created. Furthermore, the network can add new clusters without affecting the storage or recall capabilities for clusters already created and trained. When a specific input is not sufficiently close to an existing cluster (based on a vigilance parameter), a new cluster is constructed. Rare inputs will have less of an effect on the training than more frequent inputs. This type of network is primarily used for pattern classification and categorization of inputs [Jain et al., 1996]. One problem with the

Carpenter/Grossberg network is that they operate quite well when no noise is present, but often there is an excess of clusters created when the inputs are noisy [Lippmann, 1991].

4.6 Supervised learning and Feedback Recall

This class of neural networks is not as heavily used as supervised feedforward recall or unsupervised feedback recall and has fewer standard configurations. Typically these networks can be used for tasks which are suitable for unsupervised feedback recall networks. The trade-off acquired with the supervised learning method versus the unsupervised method is that typically the the storage capacity of the supervised network is increased over unsupervised recall networks, but this comes at the expense of increased training times. A few representative examples from this group is the Brain-State-in-a-Box and Fuzzy Cognitive Map networks [Simpson, 1990].

5 Phase I Technical Objectives and Results

This section reports on the technical objectives and progress performed in the Phase I grant. The technical objectives of our Phase I award were five-fold and repeated in *italics* from the Phase I proposal. The research performed toward each objective is summarized under each objective.

Compute stricter bounds on the complexity of real program state spaces. *The upper-bound on the* theoretical *complexity of our detection algorithm for an arbitrary program is probably far higher than we would expect to find in actual programs. We intend to determine a better estimated upper-bound for the complexity of the state-spaces of real programs. This research objectively encompasses determining what sort of real programs we should focus our effort on.*

Complexity of programs is a factor in this research because we are attempting to classify the behavior and usage of programs which can have a very large state space. The program state domain, $\mathcal{D}$, consists of all possible values of each component of the program state at all instructions. For each input symbol to a program, we have as many possible program executions as there are symbols in I, and, in general, for an n-step program execution sequence, we have $|I|^n$ possible executions. Assuming there are m bits of state information and that input symbols require k bits to represent, we require $m(2^{k^n})$ bits of state information. For any non-trivial input set and program, this is enormous. For example, for a program with only 1 byte of state information and inputs of at most 1 byte each executing only 50 instructions requires over 1 googol (10^{100}) bytes of storage to characterize.

To address the complexity issue, we took two approaches to examining the feasibility of the research. First, since the theoretical complexity for capturing the complete program state is too great to be a tractable, we instead decided to use only a portion of the total program state space for analysis. The next objective summarizes the heuristics used for selecting a portion of the total program state space. The second approach was to determine which programs were feasible for analysis. Our approach

was to analyze programs that met two practical criteria: first, they must be security-critical and second, the size of the program in source lines of code (SLOC) must not be greater than can be instrumented by our prototype. Fortunately, several commonly available programs meet this criteria. A range of system utilities (such as `lpr`) meet the criteria of being security critical. That is, an attack against the program can potentially compromise system security through obtaining super user access. Second, system utilities tend to be small in terms of SLOC metrics. Some network daemons, such as `fingerd`, also meet both categories and are also good targets for analysis.

Determine what state variables to track. *There are many possible state-variables that could be used to develop "legitimate-behavior" profiles. Some of those mentioned by Anderson [Anderson, 1980] include CPU time, system calls made, and I/O ranges. Choice of state-variables to track is a critical decision that will strongly affect the result of our efforts. Care must be made in determining what information to consider.*

The approach taken was to select a portion of the program state space whose execution probability is known to be invariant to input sequences. That is, the program states selected had a very high probability of being executed regardless of which inputs were run against the program. Execution analysis provides a good metric for identifying which program states are executed most often for the set of inputs run against the program.

A second approach is to identify which program states are most affected by simulated attacks against programs. This approach was used in the analysis of the `lpr` program. Since this program was known to be vulnerable to buffer overflows, the program states most effected by the buffer overflow attack were captured and used to train the neural network. The results from this approach are mixed. While the approach yielded a high percentage of correct detection of malicious behavior, the results were no better than observing solely the external inputs.

Other approaches may be viable as well for addressing the complexity of the program state space problem. Using propagation analysis, for example, to simulate program state corruptions in a program, and then determine to what extent the infections propagate to outputs may be a good metric for determining which program states are most dangerous. Finally, using a composite metric of propagation and execution analysis or using domain-to-range ratio (DRR) analysis may also be prove to be useful for determining which program states to capture given the complexity of the program state space. These approaches are proposed in Section 9: Open Research Issues.

Analyze the feasibility of building a tool. *We would like the ultimate result of this research effort to include a prototype tool suitable for further commercialization. A central objective of Phase I is to determine the feasibility of this goal. This task also encompasses researching related work in intrusion detection and program auditing.*

Based on our approaches in Objectives 1 and 2 for scaling down the complexity of the problem to a manageable task, we developed a prototype to conduct the analysis. The prototype was applied to Web application programs as well as an operating system utility. The results from the analysis (presented later in this report) indicate the

usefulness of the approaches summarized above in Objectives 1 and 2. The results also indicate that alternative heuristics for determining which program states to capture might yield even better results.

Experiment with algorithms for program state comparison. *Also critical to the project is research into advanced methods for anomaly detection for state-space values generated during run-time. Phase I will focus on two methods, with an eye towards expanding this line of inquiry in the future.*

The analysis involved experimenting with different neural network algorithms, using different parameters, and using different programs on which to perform the analysis. The two types of neural networks implemented are a back propagation network and a clustering network. In addition, several different experiments were conducted on using different combinations of internal state as well as external input to train the neural networks in order to compare the effectiveness of capturing internal states as well as external inputs. The results from the experimentation are presented in Section 7

Produce a technical report. *As a deliverable of our Phase I research, we will prepare a technical report discussing our research findings. This report will be made available to the security research community via our Website.*

A technical report has been prepared that discusses our most significant research findings. This report is now available on our Web site at http://www.rstcorp.com/papers.html. This final report will also serve as a technical report of our findings.

6 Employing Neural Networks for Anomaly Detection

Desiring the ability to detect novel or unknown intrusions, the anomaly detection paradigm was adapted for this research using neural networks. Neural networks are applied in a novel application of anomaly detection. The defining aspect of this approach is that anomaly detection is performed at the program application level, rather than the system, operating system, or individual user level.

Monitoring at the program-level adds a layer of abstraction such that abnormal process behavior can be detected irrespective of individual users' behavior. This layer abstracts out users' individual behavior and allows anomaly detection against the set of all users' behavior. The approach taken by this project was developed both as a black-box, and a white-box technology. Thus, if the source code of the monitored program is not readily available, it does not preclude the use of the proposed technique. The advantage of the black-box approach is that it can be used to detect anomalous events, *e.g.*, malicious attacks, against commercial software where the source code is unavailable. The added advantage of using a white-box capable approach is that it allows access to internal states of a program which provides additional information in detecting anomalous program behavior. Differentiation between anomalous and normal behavior will occur by creating expectations about a program's input/output behavior, as well as internal states, and then, attempting to detect deviations from the normal behavior.

An architecture of the system for analyzing programs for malicious behavior is shown in Figure 1. The system was designed to enable different applications, neural networks, and

Figure 1: An architecture for analyzing programs for anomalous or malicious behavior using neural networks.

test vectors to be used interchangeably. Thus, the architecture was constructed such that any of these components can be extracted and replaced with a different incarnation without affecting the other components.

Neural networks were first constructed and trained; then they were used to classify normal and anomalous behavior of various programs. The neural networks classified inputs, internal states, and outputs of programs as either anomalous or normal behavior. In applying this approach, it is assumed that anomalous inputs, internal states, and outputs have some correlation to malicious behavior exhibited by the program. Implementing this approach with a neural network allowed an anomaly detection approach to be taken without the need of a specific detailed statistical model or analysis. This approach has proven to be easily modifiable to monitor other processes than those already tested. One foreseeable drawback in using neural networks to classify anomalous behavior is the training period of the neural network which may take on the order of hours or days to complete. It has also not yet been determined what the effect different training sets would have on the results presented in this paper. Lastly, if new data is added to the training set, the neural network has to be re-trained over the entire training set, instead of just the data that was added to the training set.

6.1 Backpropagation Network

The backpropagation network, or backprop, is probably the most commonly used neural network. The standard architecture consists of an input layer, at least one hidden layer (neurons that are not directly connected to the input or output nodes), and an output layer. Typically, there are no connections between neurons in the same layer or to a previous layer.

Backprops have generalized capability. As such, they can produce nearly correct outputs for inputs that were not used in the training set. In theory, no more than two hidden layers are needed in a neural network since the network can generate arbitrarily complex regions in the state space [Lippmann, 1991]. One of the backprop's main drawbacks, though, is that it tends to be very computationally complex and is time consuming to train. Backprops are well-suited for applications in classification, function approximation, and prediction [Jain et al., 1996].

The training cycle of a backprop proceeds in two distinct phases. First, the input is submitted to the network and the activations for each level of neurons is cascaded forward. In the training phase, the desired output is compared with the network's output. If these vectors do not agree, the network updates the weights starting at the output neurons. Then, the change in weights is calculated for the previous layer. This process continues to cascade through the layers of neurons toward the input neurons, hence the name backpropagation.

6.2 Neural Network Implementation

Initially, the neural network constructed was a backprop network, although different types of networks may be explored in the future. The backprop implementation provided many advantages and is also well suited for this project. Backprop networks are very good at classifying complex relationships, which is precisely the goal with the proposed anomalous behavior detection.

The architecture of the backprop neural network implemented in this project is shown in Figure 2. The input layer of the network governs the number of inputs and internal states that the network uses in classification. Likewise, the output nodes govern the total number of classes the network is classifying. The backprop is trained in a supervised manner, thus the desired outputs for each input pattern must be supplied to the network during the training phase.

7 Experimental Results

This section presents the findings of applying neural networks to detecting anomalous behavior and usage of programs. The objective of the experimentation described in this section is to determine how effective the implemented neural network is at detecting anomalous use and behavior of programs. The approach examines program inputs as well as internal states of the analyzed programs to determine if the program is being used or is behaving in an anomalous manner.

Results from three experiments are presented in this report. The first experiment was a proof-of-concept experiment to determine if the neural network can potentially detect anomalous use of a Web server via input to CGI programs. The experiment simulated illegal

Figure 2: Topology of the elementary backpropagation network architecture with one hidden layer, **n** input nodes, and **q** output nodes.

access attempts to a CGI program through an Internet Web server. The neural network was trained to distinguish legitimate accesses from anomalous accesses which were attempts at perverting the system. Following training, the neural network was used to determine if anomalous requests from Web clients were detected by the prototype tool.

The second experiment implemented a clustering network to determine the viability of this network in detecting misuse of a Web server.

The third experiment was designed to detect potential misuse as well as anomalous behavior in system programs, such as `lpr`, to determine how effective a neural network can be in distinguishing inputs as normal and anomalous. The Linux `lpr` program can be subverted on specific platforms into a buffer overflow. It should also be noted that this experiment was performed in a white-box manner using internal state information. This experiment was run repeatedly using different initial weightings of the neural network in order to account for potential statistical out-liers in the results.

7.1 CGI-bin Anomaly Detection Experimental Results

7.1.1 Introduction

This section investigates the feasibility of using a neural network network to detect anomalous `/cgi-bin/` activity in real-time. CGI scripts are among the most commonly exploited Web applications used to gain unauthorized access (sometimes as the super user) to networked sites. The Computer Emergency Response Team at the Software Engineering Institute of Carnegie Mellon University has released three different CERT alerts, [CA-96.06], [CA-97.03], and [CA-97.07] on this problem (see http://www.cert.org). Once trained, the neural network runs as a daemon process and is totally transparent to both the client and also the Web server. A pre-processor parses the Web server's access log, extracts all accesses with paths including the `/cgi-bin/` directory and encodes them for the neural network to classify. The neural network is used to classify the inputs to `/cgi-bin/` scripts accessed through the Web server as either normal or anomalous.

7.1.2 Description

The neural network preprocessor and encoder work directly from the access log that the Web server produces. The preprocessor first extracts all cgi-bin accesses from the log-file and then extracts the appropriate text field which is then encoded and passed to the neural network to be classified. The inputs to the neural network are constrained such that the input is a text string of no more than 50 characters. To satisfy this constraint, the preprocessor takes only the first 50 characters in the file path field of an access log. Thus, if the line:

```
intruder.rstcorp.com - - [16/Apr/1997:13:22:28 -0400] "GET
/cgi-bin/query?name=nobody&company=&address=&address2=&city=nowhere
HTTP/1.0" 200 472
```

was in the access file, only the string:

```
/cgi-bin/query?name=nobody&company=&address=&addr
```

would be used (in an encoded form) as the input to the neural network. The encoding scheme uses the analog value of 1.0 / (ASCII value of character) for each input node.

The architecture of the neural network was a backpropagation network consisting of 50 input nodes, 100 hidden nodes, and 1 output node. The decoding scheme simply used a thresholding function to convert the real-valued output of the network to a 1 signaling an anomalous input, and a 0 to signal a normal input.

The neural network was trained with a set of 40 patterns. The ratio of anomalous to normal inputs used to train the network was 1 to 3. Ten of these patterns were anomalous inputs and comprised actual attempted exploits on the RST Web server. The remaining 30 input patterns were normal input patterns found by sampling RST's access log.

7.1.3 Results

The Web server access log-file was used as the input to the trained neural network. A total of 1,471 `/cgi-bin/` accesses via the Web server were used as input to the neural network. The total number of anomalous accesses (verified via inspection of the access logs) was 90, the remaining of the 1,381 accesses being classified as normal inputs. The neural network correctly classified 54 of the anomalous accesses (40% error rate) and correctly classified 1,380 of the normal accesses (0.07% error rate). The number of false negatives (missed intrusion attempts) were thus 36, and the number of false positives (incorrectly classified as anomalous attempts) was 1. It should also be noted that not all of the 90 anomalous accesses had a malicious nature. Many were simply CGI accesses with random characters as parameters.

It is interesting to take a closer look at the anomalous inputs that the network did not correctly classify. The first thing to note was the very constrained nature of the training set. The training set's anomalous inputs were all very similar, and thus did not represent many anomalous input patterns in general well. Furthermore, the training set was very structured in the sense that the exploits tried to access `/etc/passwd`, while once again in the recall set any file or method of exploit could be employed.

The following set of accesses represents some of the worse misclassified patterns:

```
/cgi-bin/php.cgi?/etc/yp/passwd
/cgi-bin/campas?%0acat%0a/etc/passwd%0a
/cgi-bin/echo?Hi+James%20/bin/cat%0a/etc/passwd
```

Obviously, this set of patterns represent very malicious attempts at exploiting the system and have a high desirability to be detected. The training set was composed of very similar strings but they all tried to perform the exploits through the "test-cgi" script, while the above input patterns attempted exploits through various other scripts. The neural network was not effective in detecting these intrusion attempts that did not use the "test-cgi" script.

7.1.4 Areas for Improvement

First and foremost, a larger and more varied training set needs to be constructed. This alone should vastly improve the performance of the neural network. It may also be beneficial to use a larger network, consisting of both more input nodes, hidden nodes, and possibly even add another hidden layer to the network. Other varied encoding schemes may prove to have a better result. Future experimentation will use a more diverse training set to determine the impact on detection.

7.2 Detecting Anomalous Web Accesses

This experiment attempted to extract information from a web server's access log by categorizing the accesses into clusters. Ideally, the access log would be divided up into clusters of similar accesses by some distinguishable criteria. In order to cluster the access log a Kohonen neural network was employed [Zurada, 1995]. The Kohonen network only used the IP address and file accessed fields from the access log in determining the representative clusters.

7.2.1 Description

The access log used for this experiment was the log file created by RST's Internet web server. Only GET accesses were used as input to the neural network, and at the time the log file was used it contained 20,015 GET accesses. The encoding procedure only considered the last two places in the IP address and the file path name being accessed. Thus, for the log file line:

```
epoisse.tamu.edu - - [01/Apr/1997:00:00:09 -0500]
"GET /pics/ieee-cs.gif HTTP/1.0" 200 1505
```

the fields used for encoding would consist only of:

```
tamu.edu                    /pics/ieee-cs.gif
```

The encoding procedure assigned a corresponding number to each portion of the above fields. The number would uniquely identify the access and was constructed by assigning new numbers to strings which have not been previously seen for each respective component. The path names were limited to a depth of 7 levels, and unused fields were padded by 0's.

```
IP ID 1 | IP ID 2 | dir/file name 1 | ... | dir/file name 7
```

Thus, if the above line was the first line in the access file it would be encoded as:

```
1 1 1 1 0 0 0 0 0
```

Also if the fields used for encoding of the following line in the access log were:

```
nd.edu                      /pics/webweek.gif
```

it would be encoded as:

```
2 1 1 2 0 0 0 0 0
```

The Kohonen network's inputs must be in the range $[0, 1]$, therefore the above encodings were each divided by the maximum number used in each of the respective fields before used as input activations to the neural network.

The network consisted of 9 input nodes, and 50 output nodes (each of which represented a single cluster). The number of output nodes was set artificially high to promote cluster formation (in general the larger the number of output nodes the greater chance the Kohonen network will converge with a greater number of clusters). The network was only trained for 150 epochs which is quite low for this type of network, but necessary for the network to finish in a reasonable time as the number of patterns used was so high. Typically, a Kohonen network would be trained between 500 and 10,000 epochs. Further training on a given set usually will just refine cluster definition and will often decrease the total number of clusters which the input patterns are grouped into. It would be quite rare for additional clusters to be formed after the first few epochs given the nature of the Kohonen training algorithm.

7.2.2 Discussion

The Kohonen network has a few very severe limitations, the largest being that it will only create linearly separable clusters. Thus, more complex relationships are not captured by this network. The network will essentially create hyper-planes in the state space which pass through the origin and bisect lines drawn through the cluster's center of gravity. The number of clusters and their representative vector representation in the network is also highly dependent on the initial values assigned to the weights before training.

7.2.3 Results

The network created 20 clusters of various sizes. Several clusters only contained a few input patterns while others contained thousands. Trying to extract information about how the clusters were created is very difficult (this stems from the more general problem of rule extraction from a neural network). It is also difficult to determine which fields dominated the clustering process. Thus, it is not easy to say that cluster 5 contains accesses of such-and-such from IP addresses such-and-such. The clustering process seems to be somewhat directed by the IP address more so than the files accessed, once again the specific criteria is difficult to describe.

7.2.4 Summary

The clustering process has indeed been shown to work, even on a such a large input file. However, using the resulting cluster information for any sort of security analysis seems to be of fairly minimal value. In theory, a clustering network could cluster the input patterns, an operator could then determine which groups represented normal or anomalous inputs and these results could be used as the inputs to train a supervised network. The Kohonen network does not seem to lend itself to this process though, especially for the Web server log example.

The Kohonen network should not be instantly discarded as useless for this purpose, though. It is very sensitive to both the encoding scheme and also the initial value of the weights in the network. Providing a better scheme or set of either of these may dramatically increase the reasonableness of the clusters the Kohonen network produces. The particular encoding scheme used was heavily biased towards the IP address. This is a direct result from the fact that there were much fewer endings to IP addresses than there were files that were accessed. Thus, the granularity was higher in the encoding process for the IP field dimensions, possibly allowing for easier distinction by the network. For example, let's say that the maximum number of IP ID was 100 and the maximum number of dir/file name was 1,000. Thus, the IP fields would each be separated by at least 0.01 while the filenames would only be separated by at least .001. This would make it more likely to find distinctions in the IP fields than the filenames because on average the points will be further separated in this dimension.

7.3 Detecting Anomalous Program Behavior in Linus lpr

This set of experiments explores the use of a neural network to detect anomalous behavior in the standard Linux `lpr` program. The linux `lpr` program is known to have a buffer overflow vulnerability which can be exploited through input. This exploit uses a buffer overflow attack on specific `lpr` flags to enable the user to execute a root shell or perform other root-privileged commands. The goal of this experimentation is to determine how effective the neural network is in detecting anomalous behavior in system programs due to malicious attack. To this end, the neural network was trained with malicious, normal, and in some cases, random input. This set of experiments also examines the use of internal states in the process of anomaly detection.

7.3.1 General Description

The `lpr` program is exploited in the following manner. An auxiliary program was written which filled a large character buffer. At the end of this buffer, specific character strings were added to overwrite the return address of the stack frame with the address of a new instruction to be executed, which was placed elsewhere in the buffer (this instruction was usually a /bin/sh). The program would then `exec` an `lpr` child process with this buffer as one of the flag arguments. During the experimentation, both this auxiliary program and the `lpr` program were modified to write specific inputs and internal states to a file to be used as inputs to the neural network.

The neural network's input patterns consisted of a combination of inputs from two distinct sources. The first source was the character buffer passed to the `lpr` program as a flag argument. To encode the character buffer, only the last 75 positions were used. This was done in an attempt to reduce the difference between normal flag input, on the order of say 10-30 characters, and an overflow buffer of approximately 4,000 characters. Thus, for the overflow attempts, only the last 75 of the 4,096 characters were used as inputs, and for non-overflow attempts, the entire string was used (unless of course it exceeded 75 characters). To encode these 75 input characters, they were given the value: (integer value of character) / 128.0, and 0.0 was used to pad any of the unused inputs.

The second source of input was the `len` variable in the `lpr` card() function. The card() function was chosen, because most of the `lpr` flag inputs are funneled through this particular procedure. The `len` variable is especially illustrative because it represents the length of the input parameter. The `len` variable from card() constituted an additional 8 inputs (since the procedure could be executed multiple times), and was encoded by: (`len` value)/1032, where 0.0 was used to pad any of the unused inputs.

7.3.2 The Training Sets

The input patterns for the neural networks' training sets were constructed from the following sets: anomalous inputs, normal `lpr` inputs, and random inputs. Anomalous inputs were generated using the exploit program mentioned above. The anomalous inputs did not necessarily generate a /bin/sh, but were generated to look very similar. The normal `lpr` inputs were generated by printing valid files, with various flag options. The random inputs were

generated using the fuzz program [Miller et al., 1995] were limited to a length of 80 (only 75 characters could be used).

7.3.3 The Recall Set

To test the performance of each of the neural networks trained below, one recall set was defined to test across all of the networks. The recall set was comprised of 150 normal `lpr` inputs, 50 anomalous `lpr` inputs, and 50 random inputs. *All of these input patterns were unique from those used in the training sets.* The recall sets for the different networks were only modified in the experiments to include the specific subset of input patterns that were necessary for a particular network. Hence, if the internal state was not used to train the network, then the inputs corresponding to the `len` variable would also not be clamped to the network during the recall phase.

7.3.4 The Experiments

The neural network used in the `lpr` exploit experiments was a 3 layer backpropagation network whose architecture consisted of a hidden layer, an input layer, and an output layer. There were 125 nodes in the hidden layer, 1 output node, and a variable number of input nodes.

Table 1 summarizes the experimental setup for the six experiments. The inputs to the neural network (NN) are distinguished between the experiments by whether the inputs were external `lpr` inputs or internal `lpr` states. External inputs represent standard input to the `lpr` program. If the `len` variable was used as an input to the neural network, then the Internal column is checked in the table. The number of input nodes, hidden nodes, and output nodes for each of the neural network experiments is also given in the table. The other distinguishing parameter between the experiments is whether random input was used to train the network. If random data was used in training the network, then this column is checked in the table. The neural network was trained to classify random data as anomalous, since the network attempts to distinguish between normal use and anomalous use of a program. The goal of the experimentation is to determine which of these parameters are most useful (or conversely least useful) for detection of anomalous use of a program.

	NN Inputs		NN Layers			
Num	**external**	**internal**	**Inputs**	**Hidden**	**Outputs**	**Random Data Included**
1	x		75	125	1	
2	x	x	83	125	1	
3	x		75	125	1	x
4	x	x	83	125	1	x
5		x	8	125	1	
6		x	8	125	1	x

Table 1: Experimental setup for detection of anomalous use of `lpr` program.

7.3.5 Discussion of Results

Table 2 shows the results from all six experiments. All experiments were run 30 times using 30 different initial weights of the network. The performance of the network is evaluated based on the percentage of the inputs it classified correctly, the percentage of false positives, and the percentage of false negatives. A false positive is defined as a normal input that was classified as anomalous by the network. Either of these two distinctions would constitute an error in classification by the network. A false negative is defined as an anomalous input that was classified as a normal by the network. The results are presented as averages, minima, and maxima over these runs in these categories.

To summarize the results, the best results were obtained in Experiments 3 and 4 when the neural network was trained with random data generation. Recall, that the neural network was trained to classify random data as anomalous. Including the internal state variable `len` in addition to the external `lpr` inputs did not impact the results significantly. In fact, in Experiments 5 and 6, where the neural network was trained and tested on the internal variable exclusively, the results were the weakest. The neural network in Experiment 6 did not converge to an acceptable mean squared error, so its results were omitted. In Experiment 5, the error rate was approximately 20%—exclusively false negatives. In none of the experiments were false positives detected. Since the training was heavily biased with normal inputs, in no cases did the neural network incorrectly classify a normal input as an anomalous input (the false positive case).

As a simple benchmark, consider a monkey instead of a neural network that is choosing whether the input is normal or anomalous. The monkey uses a simple algorithm for determining whether the input is normal or anomalous—he flips a coin. The likelihood that an input will be classified as normal is 50%, as is the likelihood that an input will be classified as anomalous. The actual inputs sent to the program have the following ratios: 40% were anomalous while 60% were normal. The probability that the monkey commits a false negative is the probability that an anomalous input was sent to the program and that the monkey said it was normal. Since these are independent events, this probability is 20%. The probability that the monkey commits a false positive is the probability that a normal input is sent and that the monkey said it was anomalous. This probability is calculated similarly and is 30%. The results from using the neural networks on the whole performed better than the monkey. No false positives were detected (compared to 30% for the monkey), while false negatives for the neural network ranged from 0.4% to 19.9% (compared to 20% for the monkey). For some experiments, the neural network did as bad as the monkey for false negatives (these are explained in the discussion below), while in others it did extremely well compared to the monkey (*e.g.*, Experiments 3 and 4). A discussion of each of the results from each experiment is presented next.

Experiment 1 This experiment can be considered the baseline for the `lpr` exploit experiments. It demonstrates how well the neural network performed when training only on the accessible `lpr` inputs, a reasonably sized test set, and without training randomly generated input. From Table 2 we see that the baseline error rate was 20%, exclusively composed of false negatives.

	Classified Correct			False Positive			False Negative		
Num	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
1	82.5%	80.1%	88.4%	0.0%	0.0%	0.0%	17.5%	11.6%	19.9%
2	81.9%	80.1%	85.7%	0.0%	0.0%	0.0%	18.1%	14.3%	19.9%
3	99.1%	98.0%	99.6%	0.0%	0.0%	0.0%	0.9%	0.4%	2.0%
4	98.8%	97.6%	99.6%	0.0%	0.0%	0.0%	1.2%	0.4%	2.4%
5	80.2%	80.2%	80.2%	0.0%	0.0%	0.0%	19.8%	19.8%	19.8%
6									

Table 2: Results of detection from anomalous use of `lpr` program.

Experiment 2 This experiment extends the baseline experiment, (Experiment 7.3.5), by not only using the accessible inputs to the `lpr` program, but also the internal state (represented by the `len` variable in the card() function). This experiment attempts to determine if by using an internal program state whether the performance of detection can be improved. The results do not indicate any significant impact, let alone improvement. The results are too similar to Experiment 1 to be statistically significant to note a difference.

Experiment 3 The network was trained with random inputs to further diversify its notion of anomalous inputs. The idea is to expose the network to a number of different types of anomalous use, rather then strictly well-known intrusion attempts. In general, sending random data to the `lpr` program can be perceived as an attack or misuse of the program. Training with anomalous inputs may be one way of detecting novel intrusion attempts. The performance of the network was excellent in this experiment. The average error rate for this experiment was only 0.9%. We can conclude that including randomly generated patterns in the training set vastly increased the performance of the network allowing the network to correctly classify a wide range of input patterns with a very high degree of accuracy. The results indicate that an anomaly detection approach may be useful over misuse detection in detecting novel, unknown intrusion attempts—especially when trained with random data classified as anomalous.

Experiment 4 This experiment extends Experiment 3, by including the internal state variable `len`. Once again, the goal is to determine if by adding in internal state information, whether the performance of the neural network will be improved. As seen from the Table 2, adding the internal state once again did not statistically impact the results.

Experiment 5 This experiment uses the internal state exclusively as the input to the neural network during training and recall. The results were the worst of all the experiments, but not significantly worse than experiments 1 and 2. A closer look at the data revealed that on recall, all the anomalous data randomly generated were classified as normal by the network (the false negative case). This contributed to the large false negative rate for this experiment. The reason is that the randomly generated data had a length approximately close to that of the normal training set. On the other hand, the network did very well in correctly classifying all anomalous input that were buffer overflows. Using strictly the `len`

variable is effective at detecting buffer overflows, but not in detecting other potential misuses of a program.

The problem stems from the fact that the single internal variable chosen does not provide sufficient information to classify the input patterns. Inspecting the training set reveals that all of the anomalous input patterns are exactly the same, when one is only looking at the internal variable `len`. Thus, it is no surprise that the network converged to a state in which only this input vector was classified as anomalous. To alleviate this problem, a few things could be done. The first is that additional internal variables could be used. A second is to pair this internal state with inputs, as was done in the earlier experiments. This experiment reveals one of the problematic areas encountered in this research, namely, identifying the usefulness of information and also to not provide so much information as to dilute what is important to make a distinction between normal and anomalous inputs.

Experiment 6 The neural network in Experiment 6 did not converge to an acceptable mean squared error, so its results were omitted.

8 Conclusions

The research performed under this Phase I SBIR examined the feasibility of employing machine learning techniques to detect anomalous usage and behavior of software programs. One immediate application of this research involves the detection of intrusion attempts against security-critical software. Another unexplored application is detecting and recovering from corrupted program behavior in systems where survivability or reliability is imperative. That is, if a corrupted state is detected soon enough, the program state can potentially be recovered before the corrupted state propagates to discernible outputs.

The results of this study appear to be favorable towards developing neural networks that can be applied to making systems more secure and survivable. Two different types of neural networks were implemented: a three layer backprop network and a Kohonen clustering network. The backprop network produced the best performance.

The experimental analyses conducted in this Phase I applied the neural networks to security-related applications. A study of misuse of CGI programs often misused in practice demonstrated the ability of the neural networks to correctly classify over 99% of normal accesses to CGI programs. Likewise, the study showed how the training of the networks is extremely important to detecting anomalous usage of programs. Narrow training of the network resulted in a 40% error rate in classifying anomalous inputs.

The second set of experiments analyzed the usage and behavior of a system utility program, `lpr`. A backprop neural network was trained both on the external inputs to the program as well as on an internal program state. The results from the analysis once again demonstrated the importance of an adequate training set. When the network was trained with randomly generated data, the networks classified better than 98% of anomalous usage and behavior of the `lpr` program. On the other hand, training with selected inputs resulted in a performance of slightly better than 80% correct classification.

The other lesson learned from the `lpr` experiment was that the selection of internal states for training is critical to the success of the neural network in classifying anomalous

program behavior. The internal state that was chosen in this phase of research was selected through a process of code inspection and employing heuristics about which state is most likely to be affected by anomalous usage. However, this criterion is inadequate for general purpose instrumentation of program states for anomalous behavior detection. The selection of internal states for anomalous behavior detection is an open research issue which is covered in more detail in the following section.

9 Open Research Issues

The research investigated under this Phase I SBIR has demonstrated the feasibility of applying neural networks to detecting anomalous usage and behavior in programs. The research has also revealed that there are open research issues to implementing a neural-network–based approach for intrusion detection of attacks against programs. The open research issues include the selection of a neural network and its parameterization, the training of neural networks, analysis of neural networks, the selection of application programs, and finally, the selection of the internal program states. All of these open research issues will affect the performance of the machine learning algorithm in detecting anomalous misuse and behavior. The first four issues, however, can be addressed through more experimentation and analysis. The final research issue—selection of internal program states—is the most open-ended and perhaps the most important open research issue to detecting anomalous program behavior. These issues are developed next.

Selection of Neural Network Our study of connectionist approaches to machine learning investigated a number of different types of neural networks discussed in Section 4. The backpropagation network used for most of the experimental analysis uses a supervised learning and feedforward recall approach. The benefits of this approach are that the implementation is straightforward and the networks are well-understood. Other candidates for implementation are unsupervised learning and feedforward recall networks as well as unsupervised learning and feedback recall networks. The object-oriented design of our prototype makes switching in and out different neural network implementations simple. In addition, most networks have a number of parameters that can be tuned for particular applications such as the learning rates, number of nodes, inputs, and outputs. The parameterization of the networks can be fine-tuned through experimental analysis.

Training of Neural Network The training of the neural network was one of the largest factors in its performance evaluated by the correct classification of inputs and program behavior. Narrowly trained sets performed poorly compared to a broad-range of deviant training sets. The input space of most non-trivial programs is effectively infinite. However, the expected usage of the program is often constrained within well-defined bounds. The constrained bounds on expected input can be used to train a network to recognize a broad range of anomalous input.

The results from the experimental analysis backs up this point. In cases where the neural network was trained with randomly generated data (classified as anomalous during supervised learning), the neural network performed exceptionally well. In cases, where

the neural network was trained using a narrow range of *known* bad inputs (the misuse detection approach), the neural network performed better than pure chance, but not at a level that would satisfy the requirements of the application. The expense in training using a broad range of input can be a limiting factor in using random generation of data. The open research issue is to what extent (in terms of a cost/benefit analysis) will training using random data improve the performance of the neural networks.

Analysis of Neural Network Using neural networks, we were able to classify the usage and behavior of programs as normal or anomalous. The investigation during this Phase I effort, however, treated the neural network as a black-box whose behavior was deduced from examining the neural network's inputs and outputs. The next step in the research process is to analyze the behavior of the network and determine how it is actually performing its task. In the case of connectionist networks, analysis of this sort often takes a back seat to engineering. There are, however, many analysis techniques that can be profitably applied. These include looking at the hidden units in a network to determine how they are carving up a problem space (using principal component analysis and hierarchical clustering). An analysis of this sort provides important feedback that can aid future network architectural decisions. We intend to carry out such an analysis during Phase II research.

Selection of Application Programs The selection of the program for which the neural network is trained will influence the performance of the network. For well-constrained normal input, we expect the detection of anomalous behavior to be more successful. For ill-defined inputs and program states, the performance of the neural network may not be much better than pure chance.

The utility of applying this type of analysis is maximized when applied to security-critical or mission-critical software. For example, applying anomaly detection to programs which are often attacked and exploited may give a good return on investment. Detecting anomalous internal program states may also be useful for mission-critical programs, where a corruption in state can compromise the survivability of the system. Detecting the anomalous state is an essential step towards mitigating a hazard before the system is compromised. The research investigated in this Phase I is equally applicable to systems that require high survivability, such as mission critical defense systems. The open research question is to what extent this technique has general applicability to highly survivable systems. Until we apply the techniques to mission-critical systems, we will not be able to adequately address this question.

Identification of Internal States The most significant open research area involves the identification of internal states that are optimal for classification. Since instrumentation of program states and the subsequent classification are expensive processes, they must be carefully applied. That is, the set of internal states that are instrumented, captured, and classified must be limited to permit real-time performance of the classification. The research in this Phase I proposal has identified potential methods for identifying states that would be most useful for classification. These approaches are summarized next.

Propagation Analysis One approach for identifying states most beneficial for classification is to perform propagation analysis. Propagation analysis involves perturbing program state variables and determining how often a perturbed state affects a discernible program output. The more often a perturbed program state affects an output, the more important that state is to the security or survivability of the system. RST has developed propagation analysis instrumentation technology under other research grants. The technology can be used to determine which states are most likely to affect the outcome of software corruptions.

Execution Analysis A more basic approach is performing execution analysis. Execution analysis determines which statements in the program are most often executed for given test suites. This analysis can provide useful feedback for which states are potentially unimportant to classify. One hypothesis might be that statements executed least often have minimal effect on the output and therefore should not be classified. The counter hypothesis is that if statements are not being executed due to expected inputs, then perhaps these statements deserve more careful examination. The idea is that when the low probability of execution statements are executed, they may very well cause anomalous or corrupted behavior in the output.

Combined Analysis Combining execution analysis with propagation analysis may be a practical approach for narrowing the set of states to classify to a reasonable number of "important" states. That is, execution analysis can point to low probability of execution statements that require more test vectors to observe their behavior, or they can point to statements that are always on the critical path to program outputs. Combining this analysis with propagation analysis, we can determine which of those states are most likely to affect the output when corrupted.

DRR Analysis Another approach is to use Domain-to-Range Ratio (DRR) analysis [Voas and Miller, 1995]. This type of analysis statically examines the DRR of function calls. The domain is given by the cardinality of the input to a function. The range is determined by the cardinality of the function's output. The ratio gives a rough approximation of the amount of information loss that occurs during the execution of a function. When the cardinality of the domain is greater than the cardinality of function's range, then information loss occurs. For example, for functions that return a Boolean value, the cardinality of the range is 2. Inputs to a Boolean function that have a cardinality greater than 2 will experience information loss. Why is DRR useful for our analysis? With increased information loss, the likelihood of corrupted program states within the function that are invisible at the function's output interface increases. This analysis points to functions where flaws or anomalous states may be hidden.

None of these analysis techniques promises to be a silver bullet solution to identifying which states are most useful for classification, however, they do provide statistical methods for identifying potentially important states for classification. The follow-on Phase II proposal will propose a program for investigating which of these analysis (and their corresponding metrics) are most useful for classification using neural networks.

10 Commercial Potential

Intrusion detection tools are beginning to hit the mainstream of commercial security tools. Commercial tools such as Haystack's WebStalker as well as public domain tools such as Swatch are used widely for detection of potential intruders. As discussed in Section 2, these misuse detection tools are only as good as the database from which they detect patterns. Furthermore, they can only detect known intrusion types. The anomaly detection approach investigated in this Phase I program offers the potential to detect new and potentially unknown attacks against security and mission critical systems. We believe the success of this effort will make this approach a viable commercial tool. The presence of competitors in the market only serves to grow the market while early adopters are trying out different approaches. With RST's proven commercial technology transfer program, this research has a strong chance of being commercialized.

References

[Anderson, 1980] Anderson, J. (1980). Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PN.

[D'haeseleer et al., 1996] D'haeseleer, P., Forrest, S., and Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Security and Privacy.*

[Garvey and Lunt, 1991] Garvey, T. and Lunt, T. (1991). Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference.*

[Ilgun, 1992] Ilgun, K. (1992). Ustat: A real-time intrusion detection system for unix. Master's thesis, Computer Science Dept, UCSB.

[Jain et al., 1996] Jain, A., Mao, J., and Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31–33.

[Kumar and Spafford, 1996] Kumar, S. and Spafford, E. (1996). A pattern matching model for misuse intrusion detection. The COAST Project, Purdue University.

[Lippmann, 1991] Lippmann, R. (1991). *An Introduction to Computing with Neural Nets*, chapter Part 1, pages 5–23. IEEE Press, Piscataway, NJ. in Neural Networks Theoretical Foundations and Analysis.

[Lunt, 1990] Lunt, T. (1990). Ides: an intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures.* Rome, Italy.

[Lunt, 1993] Lunt, T. (1993). A survey of intrusion detection techniques. *Computers and Security*, 12:405–418.

[Lunt and Jagannathan, 1988] Lunt, T. and Jagannathan, R. (1988). A prototype real-time intrusion-detection system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy.*

[Lunt et al., 1992] Lunt, T., Tamaru, A., Gilham, F., Jagannthan, R., Jalali, C., Javitz, H., Valdos, A., Neumann, P., and Garvey, T. (1992). A real-time intrusion-detection expert system (ides). Technical Report, Computer Science Laboratory, SRI Internationnal.

[Miller et al., 1995] Miller, B., Koski, D., Lee, C., Maganty, V., Murthy, R., Natarajan, A., and Steidl, J. (1995). Fuzz revisted: A re-examination of the reliability of UNIX utilities and services. Technical report, University of Wisconsin, Computer Sciences Dept.

[Pal and Srimani, 1996] Pal, S. and Srimani, P. (1996). Neurocomputing: Motivation, models, and hybridization. *IEEE Computer*, 29(3):24–26.

[Porras and Kemmerer, 1992] Porras, P. and Kemmerer, R. (1992). Penetration state transition analysis - a rule-based intrusion detection approach. In *Eighth Annual Computer Security Applications Conference*, pages 220–229. IEEE Computer Society Press.

[Simpson, 1990] Simpson, P. (1990). *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations.* Pergamon Press, Elmsford, N.Y.

[Voas and Miller, 1995] Voas, J. and Miller, K. (1995). Software testability: The new verification. *IEEE Software*, 12(3):17–28.

[Zurada, 1992] Zurada, J. (1992). *Introduction to Artificial Neural Systems.* PWS Publishing Co., Boston, MA.

[Zurada, 1995] Zurada, J. M. (1995). *Artificial Neural Systems.* PWS Publishing Co, Boston, MA.